# CODEFLOW Debugging Demystified!

*When your program contains a bug, it is likely that somewhere in the code, a condition that you believe to be true, is actually false. Finding your bug is a process of wading through reams of code, proving what you believe is true, is false. Here is an article focusing on the development of a simple debugging tool called CodeFlow.*

Whether you are debugging a program, fixing a bug or adding a new feature, the first thing that you may need to do is to understand the flow of the existing program. It becomes even more complex if the program you are trying to understand was developed by someone else and not by you. Here are some possible ways that one can go about trying to understand the program flow.

- Manual code walk
- Using a debugger or a profiler

But both these methods are either tedious to use or time consuming. For example, if your program has thousands and thousands of lines and is split across multiple files, then the manual code walk approach may take several man days. On the other hand, executing the program under a debugger or profiler requires some prior knowledge (for example to set breakpoints, etc) about the program to be debugged, which may not always be the case. So you definitely need some simple mechanism in addition to the ones listed here to understand the program flow from start to end.

In this article we will explore how one could develop a simple, lightweight tool called CodeFlow that can be used to trace the programs using the instrumentation option provided by the GNU gcc compiler.

When you compile a program using the GNU gcc compiler with *-finstrument-functions* option enabled, then the compiler will call *__cyg_profile_func_enter* and *__cyg_profile_func_exit* profiling functions, just after and before every function entry and exit, during runtime. These two profiling functions will be called with the address of the current function as the argument. We will use this technique to get the code flow of the programs. Listings 1 and 2 show the implementation of our CodeFlow tool.

**Listing 1: CodeFlow header file**

```
/*
*
* codeflow.h – Header file for CodeFlow for C/C++
programs
*
* Raja R.K (rajark_hcl@yahoo.co.in)
*
*/

#include <stdio.h>
#include <stdlib.h>

#define __NIF__
```

```
__attribute__((__no_instrument_function__))
#define __FUNC_ENTER__ __cyg_profile_func_enter
#define __FUNC_EXIT__ __cyg_profile_func_exit
#define TRACE_FILE "trace.out"
int indent = 0;
FILE *fptr = NULL;
```

**Listing 2: CodeFlow for C/C++ programs**

```
/*
*
* codeflow.c – CodeFlow for C/C++ programs
*
* Raja R.K (rajark_hcl@yahoo.co.in)
*
*/

#include "codeflow.h"

void exit_handler()
{
    if( fptr != NULL ) {
        fclose(fptr);
        printf("\nProgram trace saved in:
%s\n",TRACE_FILE);
    }
}

void __NIF__ __FUNC_ENTER__(void *this_fn, void *call_site)
{
    int i = 0;

    if( fptr == NULL ) {
        if( (fptr=fopen(TRACE_FILE,"w")) == NULL ) {
            return;
        }

        atexit(exit_handler);
    }

    for( i=0; i<indent; i++ ) {
        fprintf(fptr,"\t");
    }
    fflush(fptr);

    fprintf(fptr,"%x\n",this_fn);
    fflush(fptr);

    indent++;
}

void __NIF__ __FUNC_EXIT__(void *this_fn, void *call_site)
{
    indent–;
}
```

The header file shown in Listing 1 defines the necessary macros. The macro *__NIF__* will instruct the compiler not to instrument functions tagged with this attribute. The macro *TRACE_FILE* specifies the name and location of the output trace file. Listing 2 shows the implementation for the two profiling functions. I have used the global variable indent to maintain the number of TAB characters to be printed in the report file (*TRACE_FILE*). The *exit_handler* registered with atexit (which will be called during program exit) will close the report file.

As I mentioned earlier, the two profiling functions *__cyg_profile_func_enter* and *__cyg_profile_func_exit* will be called (during runtime) with the function address as the argument. You can also see this in the intermediate report shown in Listing 5. The function address, which is a hexadecimal value, will not give much useful information to the user. So for the user to understand the report created by CodeFlow, we need to convert this hexadecimal value to a humanly readable form by looking at the symbol table of the program to be debugged. Linux provides a simple command line utility called nm, which can be used to get the symbol table information for the object files in ELF format. We will use this tool to convert the report with hexadecimal value into a report that is more readable by the user. The shell script shown in Listing 3 does the job.

**Listing 3: Report generation script for CodeFlow**

```
#
# codeflow_report.sh – Report generation script for CodeFlow
#
# Raja R.K (rajark_hcl@yahoo.co.in)
#
#

#!/bin/sh

TRACE_FILE="trace.out"
TRACE_TMP1_FILE="trace.out.tmp.1"
TRACE_TMP2_FILE="trace.out.tmp.2"
PRG="$1"
NM="/usr/bin/nm"
TR="/usr/bin/tr"
CUT="/bin/cut"
CP="/bin/cp"
GREP="/bin/grep"
SED="/bin/sed"
MV="/bin/mv"
RM="/bin/rm"
SYM_FILE="$PRG.sym"

$NM -g -l –defined-only $PRG | $TR -s " " "~" | $TR -s "\t" "~"
| $CUT -f1,3,4 -d"~" > $SYM_FILE

$CP $TRACE_FILE $TRACE_TMP1_FILE

cat $TRACE_FILE |
while read line
do
    SYM_NAME='$GREP "$line" "$SYM_FILE" | $CUT -f2 -d"~"'
    $SED -e s/"$line"/"$SYM_NAME"/ $TRACE_TMP1_FILE >
  $TRACE_TMP2_FILE
    $MV $TRACE_TMP2_FILE $TRACE_TMP1_FILE
done

$MV $TRACE_TMP1_FILE $TRACE_FILE
$RM -rf $SYM_FILE
echo -e "CodeFlow report saved in: $TRACE_FILE"
```

In Listing 3 you can see that I have used *nm* to get the symbol information along with the filename and line number for the functions defined in the debugged program. Then I replaced the hexadecimal function address with the symbol names (method names) in the report created by the CodeFlow tool to get a final, readable report.

Now, with the CodeFlow tool ready and in place, let us see how we could use this tool to get the code flow for the programs.

**Step 1:** Compile codeflow.c as shown below...

```
gcc -Wall -g -O -c codeflow.c
```

**Step 2:** Compile the program for which you want to get the code flow with the *-finstrument-functions* option as follows.

```
gcc -Wall -g -finstrument-functions -o <outfile> <sourcefile>
codeflow.o
```

**Step 3:** Execute your program to get *trace.out*

**Step 4:** Generate the CodeFlow report by executing *codeflow_report.sh* against your program as shown below.

```
codeflow_report.sh <your_program>
```

**NOTE** You can see that the report creation (which will happen during program execution) and the report conversion are done in two different steps.

Listing 4, 5 and 6 show the sample program (for which we will get the code flow), the intermediate report (created by the CodeFlow tool) and the final report (created by the CodeFlow report conversion tool) respectively.

**Listing 4: Program to be debugged**

```
#include <stdio.h>

void display111() {
}

void display11() {
    display111();
}

void display1() {
    display11();
}

void display21() {
    display1();
}

void display2() {
    display21();
}
int main() {
    display1();
    display2();
    display11();
    display21();
    return 0;
}
```

**Listing 5: CodeFlow intermediate report**

```
8048612
    8048579
        8048546
            8048518
 80485df
    80485ac
        8048579
            8048546
                8048518
     8048546
        8048518
     80485ac
        8048579
            8048546
                8048518
```

**Listing 6: CodeFlow final report**

```
main
    display1
        display11
            display111
    display2
        display21
            display1
                display11
                    display111
    display11
        display111
    display21
        display1
            display11
                display111
```

Listing 6 shows the final program flow (call graph) report for the sample program shown in Listing 4. You can now see how easy it is to understand the program with our simple and lightweight CodeFlow tool. Happy debugging!!! LFY

**By:** Raja R K. The author is a lead engineer working with HCL Technologies (Cisco Systems Offshore Development Centre) in Chennai. He can be contacted at: rajark_hcl@yahoo.co.in